Trinity/NERSC-8 Mini-Application and SSP Benchmark Instructions

Introduction

Application benchmarks and microbenchmarks will play a critical role in evaluation of the offered system. The Trinity/NERSC-8 benchmarks serve three purposes:

1. The application benchmarks have been carefully chosen to represent characteristics of the expected Trinity and NERSC-8 workloads, both of which consist of solving complex scientific problems using diverse computational techniques at high degrees of parallelism.

2. The benchmarks give the Offeror the opportunity to provide NERSC and ACES with concrete data associated with the performance and scalability of the proposed system on applications that NERSC and ACES consider programmatically important.

3. The benchmarks will be used as an integral part of the system acceptance test and as a measurement of performance throughout the operational lifetime of the systems.

The Trinity/NERSC-8 benchmarks comprise tests at varying levels of the benchmark hierarchy that range from system component-level tests and kernels to Mini-Applications to Full Application Benchmarks. An aggregate performance measure, called the Sustained System Performance (SSP) metric, will be calculated based on application performance. The Offeror should pay particular attention to the SSP calculation, as it is one of the key metrics for system evaluation in this procurement.

Submission Guidelines

Benchmark results (or projections including original results) for the proposed system shall be recorded in the spreadsheet provided. Note that in the supplied spreadsheet, the "Proposed System Node Count" entry refers to the value for the full, proposed system, whether benchmarked or projected. Additionally, the Offeror shall submit electronically all completed tables, benchmark source codes, compile/build scripts, output files and documentation on any code optimizations or configuration changes on a CD or similar medium. The submitted source code shall be in a form that can be readily compiled on the proposed system. Do not include object and executable files, core dump files or large binary data files in the electronic submission. An audit trail showing any changes made to the benchmark codes must be supplied and it must be sufficient for ACES and NERSC to determine that the changes made conform to the spirit of the benchmark and do not violate any specific restrictions on the various benchmark codes.

If performance projections are used, this must be clearly indicated. The Trinity/NERSC-8 consortium will be the sole judge of the validity of any projected results. The output files on which the projections are based, and a description of the projection method must be included. In addition, each system used for benchmark projections must be described. Descriptions of the benchmarking system should be included in the supplied benchmarking spreadsheet.

Run Rules

Specific run rules for each benchmark will be included with the individual benchmark source code distribution, supplying specific requirements and instructions for compiling, executing, verifying numerical correctness and reporting results for each benchmark. Benchmark performance shall only be accepted from runs that exhibit correct execution. Only software tools and libraries that will be included for general use in the proposed system as supported product offerings are permissible for building and executing the benchmarks.

Benchmark Descriptions

Microbenchmark Tests

The Microbenchmark Tests, listed in Table 1, are simple, focused tests that are easily compiled and executed. The results allow a uniform comparison of features and provide an estimation of system balance. Descriptions and requirements for each test are included in the source distribution for each microbenchmark. The results for the proposed system shall be recorded in the provided benchmarking spreadsheet. Note, not all microbenchmarks need be submitted to respond to the RFP as several are only required at system acceptance. The stage at which a response is required for a specific microbenchmark is indicated in Table 1.

Modifications to the microbenchmarks are only permissible to enable porting and correct execution on the target platform

Table 1. Lower Level Tests (Microbenchmarks)

Benchmark	Purpose	RFP Response	Acceptance Test
STREAM	Memory Bandwidth	X	X
PSNAP	OS Jitter		X
IOR	Sequential & Parallel I/O performance	X	X
MDTEST	Filesystem Metadata Server Performance	X	X

OMB	Interconnect performance	X	X
PYNAMIC	dynamic loading and dynamic libraries		X
SMB	Message passing host processor overhead	X	X
ZIATEST	MPI application launch performance		X
UPC FT	PGAS functionality and performance		X
MPIMEMU	MPI node memory usage	X	X

MiniApplication Benchmarks

The application benchmarks are a representation of the ACES and NERSC workloads and span a variety of algorithmic and scientific spaces. The list of application benchmarks is shown in Table 2. Each application contains a README file describing how to build and run each application as well as any supporting library requirements.

For each application there are two to three problem sizes to be reported: a "small" problem to be run on a single node, a large problem that is weak-scaled to run on order-1000 nodes, and an "extra large" weak-scaled problem that is run on order 10,000 nodes. All applications have a small and large problem defined. The applications for which an extra-large problem is defined are listed in Table 2. These problems shall be at least eight times (8x) larger than a node's last level cache (and scratchpad memory if applicable, e.g. GPGPU GDDR memory). It is desired that at least 50% of the proposed system's aggregate main memory (i.e. 0.5x, 500x and 5,000x single node main memory capacity respectively) be utilized by the application. Given these guidelines, it may be be necessary to modify an extra-large problem, in conjunction with ACES/NERSC, to fit a particular architecture.

For each problem size (small and large and extra large), two cases must be reported: an unoptimized "base case" that uses as-is code and an MPI-only execution model and a "highly optimized" case that allows the Offeror broad latitude to optimize code and demonstrate the best-case performance potential of the system, especially for different execution models (MPI+X). ***It is extremely important for the Offeror to provide results for each benchmark!***

Table 2: Mini Application Benchmarks

Application	Discipline	Small/Large	Extra Large
MiniFE	Finite Elements	X	X
MiniGhost	Halo Exchange	X	X
UMT	Unstructured-Mesh deterministic radiation Transport.	X	X
AMG	Algebraic Multigrid	X	X
SNAP	Discrete Ordinates Particle Transport	X	X
GTC	Plasma Physics	X	
MILC	Lattice QCD	X	
MiniDFT	Density Functional Theory	X	

Base Case

The base case requires an MPI-only execution model and limits the scope of optimization. It is necessary to provide a point of reference relative to known systems and to ensure that any proposed system can run legacy codes that use the MPI-only execution model (any existing APIs in the codes which exploit additional parallelism, e.g. OpenMP, may not be enabled). The base case will be used to understand baseline application for applications currently using only MPI and will be used to understand the potential for application performance improvement, when compared against the optimized case. In the base case, for all applications, modifications are permissible only insofar as to enable porting and correct execution on the target platform. No changes related to optimization are permissible. Library routines may be used as long as they currently exist in an Offeror's supported set of general or scientific libraries, and must be in such a set when the system is delivered. As well, the libraries must not specialize or limit the applicability of the benchmark nor violate the measurement goals of the particular benchmark. Source preprocessors, execution profile feedback optimizers, etc. are allowed as long as they are, or will be, available and supported as part of the compilation system for the delivered systems. Only publicly available and documented compiler switches shall be used. Compiler optimizations will be allowed only if they do not increase the runtime or artificially increase the delivered FLOP/s rate by performing non-useful work.

For each problem size, the vendor is free to choose the MPI concurrency and layout (e.g. affinity, hardware multithreading) that minimizes execution time. The rationale for these choices must be detailed in the response. Note that the number of MPI tasks that can be used for a particular benchmark may be constrained by any domain decomposition rules inherent in the code.

Optimized MPI+X Cases

The optimized case allows the Offeror to highlight the features and benefits of the proposed system by submitting benchmarking results obtained through a variety of optimizations. The Offeror is strongly encouraged to optimize the source code in a variety of ways including (but not limited to):

- Using a different execution model (example: MPI+(OpenMP/OpenACC/CUDA);
- Using vendor-specific hardware features to accelerate code;
- Running the benchmarks at a higher or lower concurrency, including MPI concurrency different from the Base Case;
- Optimizing processor affinities and data layouts;

Aggressive code changes that enhance performance are also permitted as long as the full capabilities of the code are maintained; the code can still pass validation tests; and the underlying purpose of the benchmark is not compromised. Changes to the source code may be made so long as the following conditions are met:

- The rationale for and relative effect on performance of any optimization is described.
- Algorithms fundamental to the program are not replaced (since replacing algorithms may result in violations of correctness or program requirements or other chosen software decisions)
- All simulation parameters such as grid size, number of particles, etc., must not be changed.
- The optimized code execution must still result in correct numerical results.
- Any code optimizations must be made available to the general user community, either through a system library or a well-documented explanation of code improvements.
- Any library routine used must currently exist in an Offeror's supported set of general or scientific libraries, or must be in such a set when the system is delivered, and must not specialize or limit the applicability of the benchmark nor violate the measurement goals of the particular benchmark.
- Source preprocessors, execution profile feedback optimizers, etc. are allowed as long as they are, or will be, available and supported as part of the compilation environment for the delivered systems.
- Only publicly available and documented compiler switches shall be used.
- Finally, the same code optimizations must be made for all runs of a benchmark. For example, one set of code optimizations may not be made for the "small" case while a different set of optimizations are made for the "large" case.

Any specific code changes and the runtime configuration used must be clearly documented with a complete audit trail and all supporting documentation included in the submission. Trinity/NERSC-8 will be the final judge of whether optimizations will be acceptable.

SSP

Before reading the following description of the SSP and how it is used in this RFP, it is strongly recommended that the Offerer read and understand its prior definition, use, and history [1] as some terms in this brief description may otherwise be confusing. The SSP is a derived measure of computational capability relevant to achievable scientific work. It shall be used to validate the system and monitor delivered performance throughout the system lifecycle [1]. The SSP is derived from an application performance figure, P_i expressed in units of TFLOP per second per node. Given a system configured with N nodes, the SSP is the geometric mean of P_i over all applications, multiplied by N. The floating-point operation count used in calculating P_i for each of the Applications have been pre-determined by Trinity/NERSC using a hardware performance counter on a single reference system, NERSC's Hopper platform, and these values may *not* be altered. The floating-point operation counts are *not* measured on the Offeror's system; only the application run-time. The reference TFLOP counts are to be found in the mini-applications section of the Excel spreadsheet supplied with this RFP. For the purposes of the SSP calculation, the Offerer must use the run times and node counts of the large, optimized (MPI+X) problems described above.

Although the calculation of SSP is substantially the same as in prior NERSC procurements (albeit with a different set of applications), one subtle difference does exist and needs further explanation. Two of the application benchmarks, UMT and AMG, utilize iterative solvers that terminate when a convergence criterion is met. For these two codes, it has been observed that changing the number of MPI ranks for a particular problem may result in a change in the number of iterations required to achieve convergence. As this phenomenon would result in an artificial gain or loss of performance, we have redefined the calculation of P_i for these two benchmarks. For these two codes only, P_i is calculated on a per-iteration basis rather than for the entire program. Thus, in addition to reporting the run times and node counts as described in each of run rules for these two codes, we also require that the number of iterations needed to reach convergence be reported. The calculation of P_i then proceeds as follows: the total amount of work (TFLOP counts) is divided by the number of reference iterations provided by NERSC / ACES, resulting in a reference amount of work per iteration. The reported benchmark time is then divided by the benchmark number of iterations required to reach convergence on the proposed system. The performance factor, P_i, is then calculated as the reference work per iteration divided by the calculated time per iteration and the reported node count. For AMG the number of iterations to use is reported in the output on the line "Iterations = ." For UMT the number of iterations to use is reported in the output on the line "cumulativeIterationCount= ."

As calculated in the manner given above, the SSP represents an "instantaneous" measure of computational capability as of the date the Offeror's application run times were measured. To represent the cumulative computational capability of a system over a specific period of time, the instantaneous SSP is integrated over that time period by multiplying the instantaneous value by the length of time.

If the period of time of interest includes either several technology phases available at different times or technology changes (say, due to software improvements), the SSP is determined for each phase and then time-averaged over the entire period

In the SSP calculation there are two places where the number of nodes occurs: one is the number of nodes used to run the application benchmark and the other is the number of nodes with which the proposed system will be configured. Note that the former must be the physical number of nodes used to run the benchmark regardless of whether some processor elements in the nodes are left vacant (or if some are oversubscribed). T***he SSP is calculated based on the total number of physical nodes used, not (necessarily) the total concurrency.*** The Offeror should determine if the SSP increases or decreases when running in an unpacked mode.

References

1. http://www.nersc.gov/research-and-development/performance-and-monitoring-tools/sustained-system-performance-ssp-benchmark

NERSC8 Capability Improvement Run Rules

The following applies to the NERSC-8 system. As described in section 3.5.5 of the technical requirements, NERSC will require the installed system to provide a nominal 8x capability improvement over NERSC's Hopper system (which has 6,384 nodes, 153,214 cores). For NERSC, capability improvement will be judged by the results of three of the benchmarks supplied for the initial RFP: GTC, MILC, and miniDFT.

For each benchmark, NERSC will provide a benchmark problem that utilizes, nominally, two-thirds (2/3) of Hopper's current compute partition. The Offeror will then scale each benchmark problem, using the guidance provided in the README's for each benchmark and duplicated below, to utilize, nominally, two-thirds of the compute partition of the installed system. The capability metric for each benchmark will then be calculated as the product of run-time speedup and the increase in problem size. The capability improvement for each benchmark will then be arithmetically averaged to yield a capability improvement metric for the installed system.

As an example, one may scale each physical dimension (nx,ny,nz) of the MILC benchmark by a factor of two, yielding an 8x increase in problem size. If running this problem on the installed system results in a run time speedup of 1.2, the overall capability improvement metric for this benchmark is 1.2*8 = 9.6.

Benchmark Guidance:

For each benchmark:

a) It is expected that the Offeror will use the MPI+X execution model and abide by the run rules for the same as described in the RFP run rules document.
b) Each benchmark's problem size can be increased as described. The number of MPI tasks and threads used to run the benchmark will be chosen by the vendor to provide the best performance. To reach the desired capability of the installed system it is expected a combination of increased problem size and increased concurrency will be required.
c) NERSC is the sole judge of the successful completion of each benchmark and the correctness of each benchmark's capability metric.
d) Any code modifications made must pass the verification tests as described in the RFP.

For convenience, we reproduce below the guidance provided in the README for each benchmark on how to increase the problem size for each of the provided capability benchmarks.

GTC:

Capability improvement runs are enabled by increasing three parameters in the input file. For the 19,200-MPI rank large case these have the values

micell=30000
mecell=30000,
npartdom=300,

To increase the size of the problem:

1) Increase npartdom. The total number of MPI ranks = 64*npartdom

2) Increase micell & mecell simultaneously. They should be equal to 100*mpartdom

For example, to increase the max MPI concurrency by a factor of 3 over the large problem nicell=mecell=90000 & npartdom=900. In this case the increase in problem size will be 3 and the capability increase will be 3 times the runtime change.

MILC:

For the capability improvement runs, you will need to scale up the large problem (which is, itself, a weak scaled version of the small problem). For the 24,576-MPI rank large problem, the size of the four-dimensional space-time lattice
is controlled by the following parameters in the input deck:

nx 64
ny 64
nz 128
nt 192

In general, to weak scale a 4x4x4x4 (nx x ny x ny x nt) problem, one begins by multiplying nt by 2, then nz, then ny, then nz so that all variables get sized accordingly in a round robing fashion. As mentioned, the large problem is a weak scaled version of the small problem (16x16x16x24). Decomposing the large problem and following the rule just mentioned, we can see that the last variable to be updated was nz. Thus, to continue scaling the large problem, your next option is to multiply ny by 2, then nx, and then nt, and then nz, and so forth.

Note that scaling the problem to a greater number of tasks may result in run-time stability issues where the code may report 'singularity error' during the second phase of the computation. It may be possible to eliminate this error by reducing the micro-canonical time step in the parameter list for the second phase (line 34 in the file benchmark_n8/large/n8_large.in). For the large benchmark, the current value of this parameter is 0.02. Testing by NERSC indicates reducing this parameter by a factor of 10

may eliminate this error without noticeably reducing the amount of work done by the code.

miniDFT:

Capability improvement measurements are enabled by increasing the number of k-points used in the large test case. The k-point grid is specified on the last two lines of large.in:

K_POINTS automatic
nk1 nk2 nk3 1 1 1

To increase the number of k-points, adjust the (integer) parameters nk1, nk2 and nk3, which determine the size of the k-point integration grid. The number of k points increases (roughly) linearly with the product nk1 * nk2 * nk3, though a significant fraction of these points are excluded due to symmetry. Grep for "number of k points" to determine the actual number of k-points used. The increase in capability for the capability improvement calculation is the increase in the number of k-points used for the large problem (1).

The rules for the capability improvement measurement are the same as the MPI+X case (D.1.b), but require 10000 MPI ranks per k-point. The -npool command line argument should be set to the num ber of k-points.

Advice to vendors: The number of k-points is printed at the beginning of a MiniDFT run, but cannot be easily counted beforehand to set -npool. A reasonable solution is to initiate a trial-run with -npool=1, determine the number of k-points, cancel the trial-run, and restart with an appropriate value for npool.